1992

Dilution in the Hopfield neural network

Victor Manuel Castillo
San Jose State University

Order Number 1348657

Dilution in the Hopfield neural network

Castillo, Victor Manuel, Jr., M.S.

San Jose State University, 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

DILUTION IN THE

HOPFIELD NEURAL NETWORK

A Thesis

Presented to

The Faculty of the Department of Physics

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Victor Manuel Castillo Jr.

May, 1992

APPROVED FOR THE DEPARTMENT OF PHYSICS

R. Dodd

Dr. Roger Dodd

Dr. Alexander Garcia

Dr. Patrick Hamill

APPROVED FOR THE UNIVERSITY

Serena M. Stanford

ABSTRACT

DILUTION IN THE HOPFIELD NEURAL NETWORK

by Victor M. Castillo

The capacity of the Hopfield Content-Addressable neural network subject to a random dilution is investigated by numerical simulations. The sum-of-outer product learning rule is used to generate the synaptic weight matrix for the storage of M random, binary patterns. Randomly selected synaptic connection are then severed while the memory is probed to determine if the original patterns are still fixed. Other dilution methods are investigated such as one that leaves a Hamiltonian cycle, and one that does not allow isolation of nodes. In general, the critical dilution as a function of the loading ratio, $\alpha \equiv M/N$, takes a sigmoid shape. The critical dilution is also a function of the network size and the sum of the effective Hamming distances between all of the fixed patterns. A noise analysis is used to derive an estimate of the critical dilution for a given loading ratio and network size.

TABLE OF CONTENTS

List of Figures

Figure

1. Introduction

Artificial neural networks are systems of hardware and software that incorporate some of the same architecture and dynamics as do biological neural networks. The study of these systems, however, gives us more than just a better understanding of biological neural systems such as the brain. They, in fact, provide us with insight into massively parallel and distributed methods of computation. In this thesis, the effect of dilution on a neural network class known as the Hopfield model is studied by computer simulation and by analytical treatment.

1.1. Outline of Thesis

Before proceeding, a road map to this thesis is given here to help tie together the sections that follow. The motivation for studying neural networks as a computational method is presented in the remainder of this section, along with a brief description of how neural networks are typically modelled. This is followed by a detailed discussion of the Hopfield model that includes a mathematical description, an example of how it is used as a computational method, and some of its limitations. In Section 3, descriptions of the various dilution methods, key parameters, and general terms used in describing the network model are provided. In Section 4, the dilution algorithms used are discussed.

In Section 5, the results of the numerical experiments on small diluted networks are presented. A predictive model based on a sigmoid fit to the numerical results is given. In Section 6, a noise-to-signal analysis is used to derive an expression that fits a special class of the results observed in the simulations. A summary is given in Section 7.

1.2. The Biological Motivation

The brain has been the subject of scientific curiosity for centuries. However, it was not known that the nervous system is made up of many interconnected cellular units until the end of the nineteenth century when Santiago Ramon y Cajal[1] used a staining technique with an optical microscope. We now know that biological nervous systems are made up of many relatively simple neurons that are connected to many other neurons by dendritic interconnections as shown in figure 1. At the interface of the dendrite of one neuron and the body of another, a chemical bridge called the synaptic interface exists. When the electrical pulse from a source neuron gets to this interface, ion transfer must occur in order for this signal to continue. The synaptic interface thus regulates the flow of information from neuron to neuron. It is widely accepted that the dynamics of this many-node system subject to the influence of these synaptic interfaces is responsible for the complex functionality of biological neural systems.

---

[1] Santiago Ramon y Cajal was a Spanish medical doctor who in 1888, used a newly developed tissue staining technique to demonstrate that neural tissue is made up of discrete cells that are separated by small gaps.

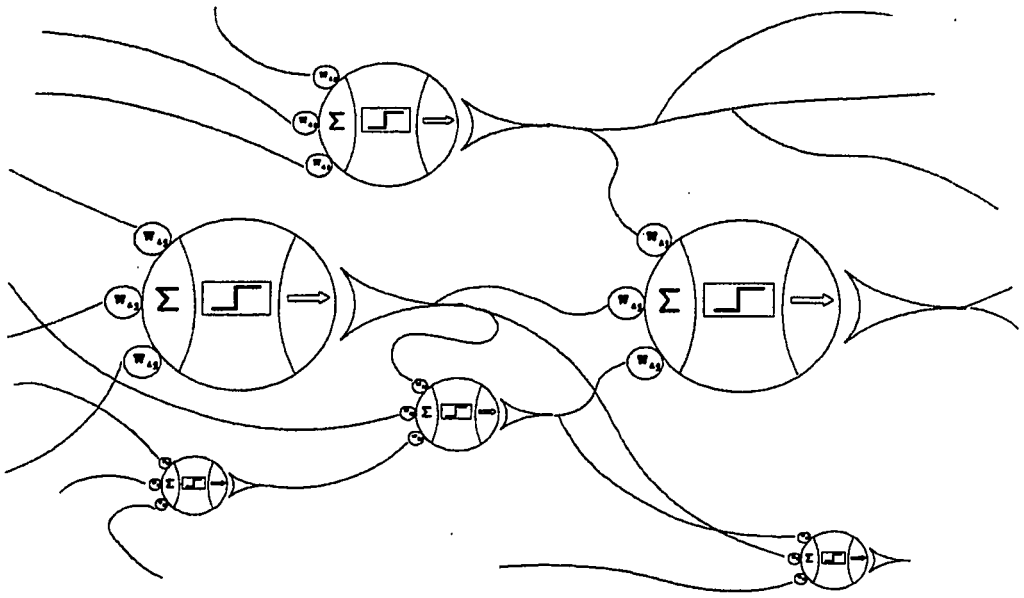

Figure 1. A diagram of a neural system. The details in this figure represent nodes, dendrites, synaptic interfaces, and some interneuron functions.

1.3. Neural Networks as a Computational Method

Artificial neural network models, in general, are highly simplified models that are only loosely based on their biological counterparts. They do however, exhibit some of the same complex processing abilities. Other models, such as models of spin glasses for example, are over-simplified but still are able to unveil complex macroscopic behavior that are not apparent from the detail of the individual elements.

The Ising model of a magnetic material uses a spin that can only be in an up or down position to represent the state of a microscopic domain of a larger system. Long-range interactions influence the dynamics of the individual spins at finite temperatures.

These interactions can have a ferromagnetic (prefer parallel states) or anti-ferromagnetic nature. The magnetization of this system can be shown to go through a phase transition as the temperature of the system reaches a critical temperature T_c (c.f. [1]).

Cellular automata models of dynamical systems assume that the system can be divided into a large number of cellular spaces that are each allowed to have a finite, discrete state. The states of the cells are allowed to evolve in time based on a set of rules and the states of its neighbors. Cellular automata (and Lattice Gas[1]) methods have been shown to model complex systems such as fluids in motion [2], chemical reactions [3], and diffusion constrained by irregular boundaries [4]. For a more complete survey of this method, see [5] or [6].

All of these models fall into the category of collective systems. Algorithms based on collective system models have certain advantages over more conventional numerical techniques. For example, the individual nodes can be made to process in parallel, thus computations can be done at speeds far greater than with serial procedures. The fact that the information is distributed over the whole system leads to a robustness or fault tolerance of the system. In addition, since the processing nodes are identical and relatively simple, a system can be constructed economically.

Algorithms that fall into the category of neural network models can perform difficult computational tasks, but execute them in a way that is very different than that

---

[1] Lattice Gas methods are similar to cellular automata methods in that a grid is established. Lattice gas methods however, treat the changing of a cellular state as a discrete particle moving about the grid.

of traditional methods. Instead of following a sequence of instructions to change the value of a variable, a neural network changes the value of a variable by "evolving" its components individually from some initial value to a desired value. To visualize this, consider the state of a system of N nodes at some time. The value of a variable can be represented by a N-bit word or vector, ξ, having the components ξ_i, where ξ_i is the state of node n_i. The computational task is to take the network that has been set to some initial state $\xi^{(s')}$ and transform it in time (through an iteration operation) to some desired state $\xi^{(s)}$. The initial state $\xi^{(s')}$ is the input of this procedure and $\xi^{(s)}$ is the desired output. Since the transformation process is an iterative operation, we also require that once the system gets to the state $\xi^{(s)}$, that it remains unchanged by the process. We then say that $\xi^{(s)}$ is a fixed state or a *fixed point* of the dynamics.

1.4. Models of Neural Systems

The accumulation of experimental data from biological neural systems led to many models of how they work. In the early 1940's, Warren McCulloch and Walter Pitts [7] developed a theory based on the "all-or-nothing" character of nervous activity. The state of a neuron is given a binary value of 1 to represent a firing neuron and 0 for a non-firing neuron. The state of the neuron changes in discrete time steps so that the new state is a function of a linear combination of the states of all the other neurons,

$$\xi_i^{(new)} = \Theta\left(\sum_j T_{ij}\xi_j^{(old)} - \theta_i\right) \quad , \tag{1}$$

where ξ_i and θ_i are the state and threshold of node n_i respectively, and Θ is some nonlinear function. The value T_{ij} represents the synaptic coupling strengths between neurons n_i and n_j. The synaptic coupling strength can be either excitatory or inhibitory depending whether its value is greater or less than the threshold. This means that the system can have both ferromagnetic and anti-ferromagnetic coupling interactions occurring simultaneously. This leads to *frustrated* states and results in complex dynamics. McCulloch noted how this resembles nervous activity but had no way to determine the coupling strength coefficients that would lead to a specific desired outcome.

Neural biologists came to find that the synaptic coupling strengths are not constant over time. Donald Hebb [8] proposed that a synaptic coupling increased in strength when the neurons that it connected were simultaneously firing. This has come to be known as the Hebbian hypothesis. Methods of determining synaptic weights that agree with this hypothesis are said to obey Hebb's Rule.

Other models were developed based on the ideas of neurons with binary states evolving in discrete time under the influence of variable synaptic coupling strengths. These include the Perceptron [9], the back-propagation model [10], and the brain-state-in-a-box [11]. In general, the models of neural networks can be described in mathematical terms [12] as a directed graph with the following properties:

1. A state variable, ξ_i, is associated with each node.
2. A real-valued weight, T_{ij}, is associated with each edge (i,j).
3. A real-valued threshold, θ_i, is associated with each node.
4. A transfer function, Θ_i, is defined for each node and determines its state at the next time step.

2. The Hopfield Model

John Hopfield [13] introduced a mathematical model of biological neural networks accompanied by analysis loosely based on that of the long-range Ising model. This model incorporates a large number of McCulloch-Pitts type nodes that are fully connected by variable synaptic interconnections that obey Hebb's rule. The individual nodes can assume only two possible states: +1 for a firing neuron and −1 for a non-firing neuron.

2.1. Hopfield Dynamics

The dynamics of the system are described by the discrete time evolution,

$$\xi_i^{(new)} = sgn\left(\sum_{\substack{j=1 \\ j\neq i}}^{N} T_{ij}\,\xi_j^{(old)}\right) \quad , \tag{2}$$

where sgn(x) = 1 for x > 0 and sgn(x) = −1 otherwise. The choice of sgn(0) is arbitrary but must be consistently applied. The updating of the state of the individual nodes may be done synchronously or in a Monte Carlo[1] fashion. Hopfield used the fact that for symmetric coupling interactions, a simple energy function exists for the system to show that the state vector will always evolve to the fixed vector closest to it. This can be

[1] This refers to the random selection of a node for updating.

shown for asynchronous operations, if we consider the Lyapounov function[1] for this system in state s,

$$E^{(s)} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N} T_{ij}\xi_i^{(s)}\xi_j^{(s)} \quad . \tag{3}$$

We can see that the change in the energy due to a single Ising spin flip of node k is

$$\begin{aligned}
\Delta E &= E^{(new)} - E^{(old)} \\
&= -\frac{1}{2}\left[\sum_{\substack{i=1 \\ i\neq k}}^{N}\sum_{\substack{j=1 \\ j\neq i,k}}^{N} T_{ij}\xi_i^{(s)}\xi_j^{(s)} + (\xi_k^{(s)}+\Delta\xi_k)\sum_{\substack{j=1 \\ j\neq k}}^{N} T_{kj}\xi_j^{(s)} + (\xi_k^{(s)}+\Delta\xi_k)\sum_{\substack{i=1 \\ i\neq k}}^{N} T_{ik}\xi_i^{(s)}\right] \\
&\quad + \frac{1}{2}\left[\sum_{\substack{i=1 \\ i\neq k}}^{N}\sum_{\substack{j=1 \\ j\neq i,k}}^{N} T_{ij}\xi_i^{(s)}\xi_j^{(s)} + \xi_k^{(s)}\sum_{\substack{j=1 \\ j\neq k}}^{N} T_{kj}\xi_j^{(s)} + \xi_k^{(s)}\sum_{\substack{i=1 \\ i\neq k}}^{N} T_{ik}\xi_i^{(s)}\right] \qquad (4) \\
&= -\Delta\xi_k \sum_{\substack{j=1 \\ j\neq i}}^{N} T_{kj}\xi_j^{(s)} \quad .
\end{aligned}$$

Using the update rule (2), we can express $\Delta\xi_k$ as,

$$\Delta\xi_k = \xi_k^{(new)} - \xi_k^{(s)} = sgn\left(\sum_{\substack{j=1 \\ j\neq k}}^{N} T_{kj}\xi_j^{(s)}\right) - \xi_k^{(s)} \quad . \tag{5}$$

[1] A Lyapounov function is a monotonously decreasing energy function of the configuration of a dynamical system.

By putting (5) into (4) we get,

$$\begin{aligned}
\Delta E &= \xi_k^{(s)} \sum_{\substack{j=1 \\ j \neq k}}^{N} T_{kj} \xi_j^{(s)} - sgn\left(\sum_{\substack{j=1 \\ j \neq k}}^{N} T_{kj} \xi_j^{(s)} \right) \sum_{\substack{j=1 \\ j \neq k}}^{N} T_{kj} \xi_j^{(s)} \\
&= \xi_k^{(s)} \sum_{\substack{j=1 \\ j \neq k}}^{N} T_{kj} \xi_j^{(s)} - \left| \sum_{\substack{j=1 \\ j \neq k}}^{N} T_{kj} \xi_j^{(s)} \right| \qquad (6) \\
&= \xi_k^{(s)} \, h_k^{(s)} - |h_k^{(s)}| \\
&= \begin{cases} 0 & \xi_k^{(s)} = sgn(h_k^{(s)}) = \xi_k^{(new)} \\ -2|h_k^{(s)}| & otherwise \end{cases} .
\end{aligned}$$

This shows that any single change due to the update rule (2) will either decrease the energy of the system or not change it. Since the energy of the system is bounded below by $-\Sigma_i \Sigma_j |T_{ij}|/2$, the system is guaranteed to monotonously converge to some fixed state. A vector ξ, is considered fixed under these dynamics if $\xi^{(new)} = \xi^{(old)}$ for all time. It is then obvious from (2) that what vectors represent local energy minima is entirely determined by the choice of synaptic weights $\{T_{ij}\}$.

2.2. Content-Addressable Memory (CAM)

Hopfield argued that a system with these dynamics would have computational capabilities. If, for example, the location of a fixed point of this system is a code that represents an important piece of information, then the information is *stored* in the system. In addition to just storing the information, this system has an error-correcting ability. If at some later

time, only a part of the information is known, the system could be set to this initial state where some of the nodes have a "correct" value (it is not necessary to know which ones) and the rest do not. The system would then evolve to the fixed state where all of the nodes have the correct value. An information system with this capability is called a *content-addressable memory* because information is retrievable based on a partial knowledge of its content. Hopfield showed that his simplistic model had the ability to store and recall a number of patterns.

2.3. Hebb-Rule Learning

The process of determining the values of the synaptic weights that establishes a desired set of patterns as fixed patterns is referred to as *learning*. This is a process that must precede the recall phase. There are, however, a large number of sets of weights possible, and for any set, there are a large number of local minima. If we wish to fix M patterns into memory, a simple but sufficient learning algorithm for the Hopfield network that supports the Hebbian hypothesis is the *sum-of-outer products* learning rule,

$$T_{ij} = \sum_{s=1}^{M} \xi_i^{(s)} \xi_j^{(s)} , \quad T_{ii} = 0 \quad . \tag{7}$$

This method supports the Hebbian synapse hypothesis because the value of T_{ij} is just the number of times that node i and node j agree minus the number of times that they disagree over all of the patterns to be fixed in memory. Once the set of weights has been

determined, they are quenched. This is unlike the way that the brain is thought to work since learning and recall seem to happen simultaneously. This, however, would be much more difficult to model.

The result of (7) is the same whether it is done in a synchronous or an asynchronous manner. The matrix $T = \{T_{ij}\}$ is the sum of outer products of all of the memory vectors $\xi^{(s)}$ that are to be fixed minus M times the NxN identity matrix I_N,

$$T = \sum_{s=1}^{M} \xi^{(s)} \xi^{(s)T} - MI_N \quad . \tag{8}$$

The result of the sum-of-outer products is a symmetric, zero-diagonal, quantized matrix of weights, $\{T_{ij}\}$.

As stated before, the synaptic weight, T_{ij}, determines how much influence the state of node n_i has on n_j under the recall dynamics of (2). In this symmetric model, n_i has the same effect on n_j that n_j has on n_i. To maximize the error correcting ability of the system, the state of the individual node at time t is not allowed to affect its state at time t + 1. Thus the diagonal of the matrix is set to zero (i.e. $T_{ii}= 0$).

2.4. A Simple Example of CAM

Before continuing, a simple example is used to demonstrate the methods discussed so far. For a small network of five nodes, we can fix two patterns (N=5, M=2), $\xi^{(1)}$ and $\xi^{(2)}$. Let

$\xi^{(1)} = \{-1,-1,-1,-1,-1\}^T$ and $\xi^{(2)} = \{-1,+1,-1,+1,-1\}^T$. Then $\mathbf{T}^{(1)} = \xi^{(1)}\,\xi^{(1)T} - \mathbf{I}_N$ and $\mathbf{T}^{(2)} = \xi^{(2)}\,\xi^{(2)T} - \mathbf{I}_N$.

$$T^{(1)} = \begin{bmatrix} 0 & +1 & +1 & +1 & +1 \\ +1 & 0 & +1 & +1 & +1 \\ +1 & +1 & 0 & +1 & +1 \\ +1 & +1 & +1 & 0 & +1 \\ +1 & +1 & +1 & +1 & 0 \end{bmatrix}, \qquad T^{(2)} = \begin{bmatrix} 0 & -1 & +1 & -1 & +1 \\ -1 & 0 & -1 & +1 & -1 \\ +1 & -1 & 0 & -1 & +1 \\ -1 & +1 & -1 & 0 & -1 \\ +1 & -1 & +1 & -1 & 0 \end{bmatrix} \tag{9}$$

Therefore, $\mathbf{T} = \mathbf{T}^{(1)} + \mathbf{T}^{(2)}$ is the matrix of weights that fixes both $\xi^{(1)}$ and $\xi^{(2)}$.

$$T = T^{(1)} + T^{(2)} = \begin{bmatrix} 0 & 0 & +2 & 0 & +2 \\ 0 & 0 & 0 & +2 & 0 \\ +2 & 0 & 0 & 0 & +2 \\ 0 & +2 & 0 & 0 & 0 \\ +2 & 0 & +2 & 0 & 0 \end{bmatrix} \tag{10}$$

Now that $\mathbf{T}$ has been determined, we say that $\xi^{(1)}$ and $\xi^{(2)}$ have been learned. To test this, we can "probe" the memory with noisy patterns $\xi^{(1')}$ and $\xi^{(2')}$ and allow the system to evolve under the dynamics in (2). Let $\xi^{(1')} = \{-1, -1, +1, -1, -1\}^T$ and $\xi^{(2')} = \{-1, +1, -1, +1, +1\}^T$. We see that $\mathbf{T}\xi^{(1')} = \{0, -2, -4, -2, 0\}^T$ and $\text{sgn}(\mathbf{T}\xi^{(1')}) = \{-1, -1, -1, -1, -1\}^T = \xi^{(1)}$. Also, $\mathbf{T}\xi^{(2')} = \{0, +2, 0, +2, -4\}^T$ and $\text{sgn}(\mathbf{T}\xi^{(2')}) = \{-1, +1, -1, +1, -1\}^T = \xi^{(2)}$. Each pattern could be recalled in just one iteration. By continuing the iteration process, the original patterns will continue to be returned.

Of course these examples are contrived in order to demonstrate the mechanics of the Hopfield sum-of-outer products content-addressable memory. In Section 2.5. the capacity limitations of this model are discussed.

2.5. The Capacity of the Hopfield Network

The number of patterns that can be stored by a neural network is related to the information capacity of the system, which is limited by the number of nodes. The *loading ratio*, $\alpha \equiv M/N$, is defined as the ratio of the number of fixed memories to the number of nodes. The maximum loading ratio of a network, α_c, is called the *critical loading ratio* or *critical capacity.*

Hopfield observed in his simulations that random patterns could be stored and recalled with a very small error as long as $\alpha < .15$. Once this critical loading ratio was exceeded, the errors in recall become severe. Amit et al. [14] used a mean-field approximation based on the replica method developed for spin-glass models to estimate the critical loading ratio at $\alpha_c = 0.138$ for the storage of random, *un*correlated patterns. This analysis, however, is not valid for zero temperature (deterministic) recall.

Kanter and Sompolinsky [15] developed a new learning algorithm called the *pseudo-inverse* rule that has a critical loading ratio of about 1. This requires that the patterns be linearly independent so that the pattern matrix $\{X_{i\upsilon} = \xi_i^{\upsilon}\}$ can be inverted. Gardner [16] carried this approach further and required that the synaptic weights be normalized. The sum of T_{ij}^2 must strictly equal N. With this constraint, a set of optimal

synaptic weights (asymmetric) can be determined such that α_c can be as great as 2. In this approach, the spins are quenched and the connection strengths are allowed to vary in a non-trivial way. The convergence theorem, however, guarantees that the optimal set of weights will be found if they exist.

McEliece [17] used a noise-to-signal analysis to estimate the critical capacity for the storage of random, uncorrelated patterns using the sum-of-outer products learning rule as,

$$\frac{M}{N} \leq \frac{1}{2 \ln N} \tag{11}$$

if we allow a few errors. This analysis is valid for deterministic recall.

2.6. The Effects of Dilution

The robustness of this information storage system is one of its greatest attributes. In particular its tolerance to synaptic damage has been the focus of some interesting work. In general, neural networks show a *graceful degradation* of memory much like the symptoms of Alzheimer's Disease.

Derrida et al. [18] show that under *extreme dilution* ($C << \ln N$, where C is the number of connections) that the case for the asymmetric connections is exactly solvable. In this case the critical storage capacity can be as high as $2/\pi$. In a related work [19], this same critical storage capacity is shown to hold for *strong dilution* ($C = O(\ln N)$).

The tolerance of this model to *weak dilution* [20] is shown to be nil for the saturated case (α approaching 2).

An estimate of the critical capacity as a function of the dilution for the symmetric connection case was made by Sompolinsky [21] based on the method used by Amit et al. in reference [14]. The approximation showed the capacity of the diluted network to be slightly greater than the naive approximation of $\alpha_c(c) = .138c$, where c is the percent of remaining connections. This analysis, however, may not be valid for zero temperature dynamics. In numerical simulations by Koscielny-Bunde [22], the critical capacity is shown to agree with this linear approximation for zero temperature retrieval. In these simulations, a Monte Carlo analysis is used on moderate size networks of 200, 400, and 800 nodes. No variation is seen among the network sizes studied.

In this thesis, small networks (N=50, 100, and 150) using the sum-of-outer product learning rule and deterministic recall dynamics are investigated by numerical simulation. An analysis is carried out using a noise analysis similar to that of McEliece.

3. Definitions

Before proceeding, some definition is given to important terms and parameters used in the sections to follow.

3.1. The Directed Graph Model

As mentioned before, a neural network model can be described in mathematical terms as a directed graph with certain properties. Terms and concepts from Graph Theory can be used to better describe some of the methods and results discussed in this study.

The nodes of a neural network can be described as the vertices v_i and the synaptic connections by the edges (i,j), $i,j \in \{1, N\}$, of a graph. The general form of a directed graph is not necessary since the connections are assumed symmetric. The edges can be given labels that are independent of the synaptic value, T_{ij}. In the dilution algorithms discussed in Section 4, the edges are allowed to have one of three labels. If an edge is labeled 0, then it is considered to be disconnected (this is equivalent to having $T_{ij} = 0$). If the edge has the label TL, then it has a temporary status and may be disconnected. If an edge has a label PL, then has a permanent status and may not be disconnected by the dilution algorithm. The matrix of all of the edge labels is called the adjacency matrix. Since the connections are symmetric, the status of (i,j) is the same as that of (j,i). Also, since the vertices are not self-connected, the diagonal of the adjacency matrix is zero. In the dilution algorithms, an edge is selected for evaluation with a uniform probability.

3.2. Dilution Methods

Three dilution methods are used in this study:

I. Edges are randomly disconnected with equal probability. This is referred to as simple dilution.

II. Same as type I, but with an additional constraint that no node is allowed to become totally isolated. This is a minimum coverage routine.

III. Same as type I, but requiring that a Hamiltonian cycle remain intact throughout the dilution.

Figure 2 shows the result of complete dilution for the methods studied. The top figure is a fully connected network. The bottom figures are the result of simple dilution, dilution to minimum coverage, and dilution to a Hamiltonian cycle (from left to right).

3.3. The Dilution Parameter

The capacity of a neural network is defined in this study as the number of memorized patterns that can be exactly recalled without error for a network of N nodes and subject to dilution d. The *dilution parameter*, d, is defined as the total number of edges that have been severed, normalized to the total number of edges in the fully connected network. The dilution parameter for a network of size N is given the value,

$$d = \frac{2(number\ of\ cuts)}{N^2-N} \quad . \tag{12}$$

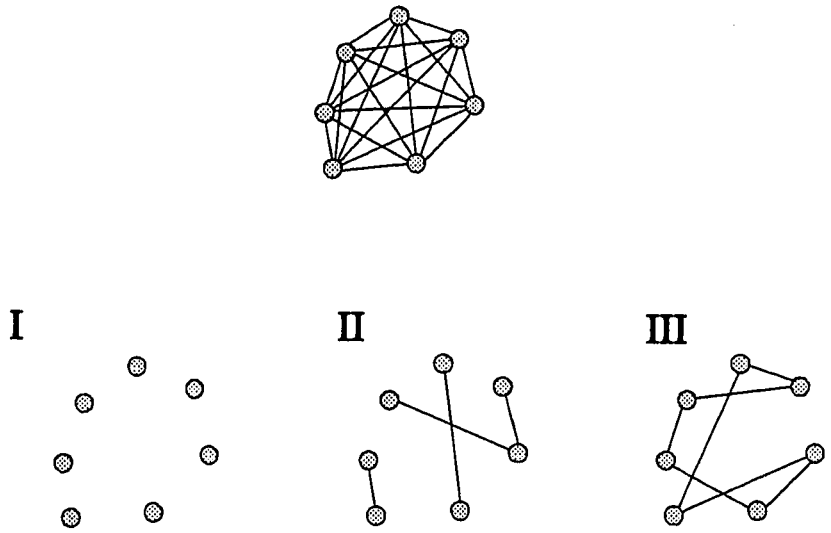

Figure 2. The result of complete dilution for the methods used. The top figure is a fully connected network. The bottom figures are the result of type I, II, and III dilution.

For type I dilution, d has a maximum value of 1 since all of the $(N^2 - N)/2$ edges can be cut. For type III dilution, the maximum value of d is

$$d_{max} = \frac{N-3}{N-1} \quad , \tag{13}$$

since a Hamiltonian cycle of N edges is left intact. The maximum value of d for type II dilution varies on the random order that the edges are removed. In this case d_{max} is greater or equal to d_{max} for type III dilution, but less than or equal to one.

3.4. Critical Dilution

In the experiments, the capacity of the network subject to random dilution is systematically examined. In particular, the value of the dilution parameter, d, at which the network fails to return all of the assigned patterns is determined. This value is referred to as the *critical dilution*, d_c. In the experiments, the critical dilution is investigated for each loading ratio, α. The plot of the critical dilution verses the loading ratio is referred to as the *critical dilution profile.*

3.5. The Effective Hamming Distance (H_E)

The state of the of the network is described as a point in N-space (or a N-dimensional vector from the origin to this point) moving under the dynamics of (2) towards one of the local minima (fixed points) established by (8). The distance between any two points in this space is defined by the *Hamming distance.* The Hamming distance is a measure of how different one binary vector is from another. The Hamming distance between the random, binary vector $S=\xi^{(1)}$ and the vector $T=\xi^{(2)}$ is the number of components that are different between them. This can be expressed mathematically as

$$H(S,T) = \frac{1}{2}\sum_{i=1}^{N} |S_i - T_i| \quad . \tag{14}$$

The Hamming distance between two binary vectors of N components has a value between 0 and N, where a Hamming distance near zero means that the vectors are much alike.

On average, the Hamming distance between any two vectors with independent, random components is N/2.

Another restriction of CAM, is that in order to establish a new fixed point, it must be a sufficient distance away from previously established fixed points. If the new pattern is too close to an existing fixed point it will disturb it and the result could be that neither of the points become local minima.

Since the outer-product learning rule (8) gives the same result for the addition of $\xi^{(\mu)}$ to the memory as it does for $-\xi^{(\mu)}$, both of these patterns become fixed points. This being so, the ability of a new pattern to be fixed depends on the Hamming distance between it and all of the previously fixed points, and the Hamming distance between it and the negative of the previously fixed points. An important quantity introduced by Dasgupta [23] is the *effective Hamming distance*,

$$H_E(S,T) = MIN[H(S,T)\ ,\ H(S,-T)]\ . \tag{15}$$

The effective Hamming distance is just the smaller of the Hamming distance between S and **T** and that of S and **–T**. Since the patterns are binary (S_i, $T_i \in \{-1,1\}$) then $2\ |\ S_i - T_i\ | = (S_i - T_i)^2$, therefore,

$$\begin{aligned} H(S,T) &= \frac{1}{4}\sum_{i=1}^{N}(S_i - T_i)^2 = \frac{1}{4}\sum_{i=1}^{N}(S_i^2 + T_i^2 - 2S_iT_I) \\ &= \frac{1}{4}\left[\ \langle S,S\rangle + \langle T,T\rangle - 2\langle S,T\rangle\ \right] \\ &= \frac{1}{2}\left[N - \langle S,T\rangle\right]\ . \end{aligned} \tag{16}$$

By the same argument,

$$H(S,-T) = \frac{1}{2}\,[N + \langle S,T\rangle] \quad , \tag{17}$$

where <S,T> is the inner product of S and T. Using (16) and (17), the effective Hamming distance can be expressed as,

$$\begin{aligned} H_E(S,T) &= \frac{1}{2} MIN\,[N - \langle S,T\rangle\ ,\ N + \langle S,T\rangle] \\ &= \frac{1}{2}\,[N + MIN(-\langle S,T\rangle\ ,\ \langle S,T\rangle)] \\ &= \frac{1}{2}\,[N - |\langle S,T\rangle|\,] \quad . \end{aligned} \tag{18}$$

4. Dilution Simulations

In this study, computer simulations are used to study the effect of dilution on finite size neural networks. The main theme of this investigation is to determine the amount of dilution a small network can tolerate for a given loading ratio. The extent of the error-correcting ability of the network is not considered in the way it has been in other studies (c.f. [21]). In addition to simple dilution, other systematic dilution methods are investigated to give further insight to the pattern storage mechanism. In this section, the details of the dilution algorithms are discussed.

In all of the computer experiments, the critical dilution profile is determined using the same general procedure. First, a single random pattern is fixed into the memory using the sum-of-outer products learning rule (8), and then verified that it is still a fixed pattern after a single edge is disconnected ($T_{ij}= T_{ji}= 0$). The dilution-verification cycle is then repeated until the network fails to return the pattern or no removable (TL) edges remained. The number of nodes N, number of patterns M, and dilution d, is then reported. Next, the synaptic weight matrix is re-initialized, another random pattern is generated, and both patterns are fixed into memory. The dilution-verification cycle is repeated for both of these patterns. The cycle stops when one of them ceases to be fixed or all TL edges have been removed. The procedure continues until the number of random patterns that are tried to be simultaneously fixed is equal to .25N. This well exceeds the theoretical critical capacity of the undiluted network.

4.1. Simple Dilution

The most straightforward way to dilute the network is to pick two nodes n_i and n_j at random, i,j $\in$ {1,N}, and remove the edge (i,j) that joins them. It is, however, questionable whether the effect of dilution would be dominated by the isolation of nodes, effectively lowering the size of the network. To address this, other methods are investigated.

4.2. Dilution to Minimum Coverage

To prevent a node from becoming isolated, an edge-labeling algorithm is used. An edge is labeled PL if a node would become isolated by its elimination. The edge-labeling algorithm is incorporated into the following dilution sequence routine:

Random Dilution to Minimum Coverage

Given a fully connected graph G = (V,E) with vertices v_i and edges (i,j) i,j $\in$ {1,N}, this algorithm randomly dilutes the graph but does not allow a node to become uncovered.

INPUT: Initial adjacency matrix.
OUTPUT: A random sequence of edges to be diluted that will leave a subgraph of the initial set of vertices having the minimum number of edges so that all of the vertices are covered.

START

1. *Vertices v_i and v_j are randomly selected.*
2. *Evaluation of edge (i,j)*
   a. If (i,j) = 0 then goto START.
   b. If (i,j) = PL then goto START.
3. *Evaluation of v_i.* If there is only one non-zero term in row i of the adjacency matrix, that edge is (i,j).

a. Set (i,j) and (j,i) to PL.
b. Goto START.

4. *Evaluation of v_f.* If there is only one non-zero term in row j of the adjacency matrix, that edge is (i,j).
   a. Set (i,j) to PL.
   b. Goto START.
5. *Dilution.* The dilution sequence list is updated and the edge is labeled 0.
   a. i and j are added to the dilution sequence list.
   b. Set (i,j) and (j,i) to 0.
6. *Check for TL.* If there is at least one TL edge in the adjacency goto START.

END

4.3. Dilution to a Hamiltonian Cycle

The previous method does not allow complete isolation of any nodes, but small clusters of about two or three nodes are formed. This may lead to some spurious phenomenon itself. In this procedure, partitioning of the graph is prevented by requiring that a Hamiltonian cycle remain intact at all times. A Hamiltonian cycle is a subgraph of all N vertices connected by a closed path except for the starting/ending node. The path also starts and ends on the same node. The implementation of this constraint requires the use of an edge labeling technique as described in the previous routine. In this routine, however, the nodes are first placed in a random order. The edges connecting the nodes in this sequence are labeled permanent before the cut order is determined. The following algorithm is used.

Dilution to a Hamiltonian Cycle

Given a fully connected graph G = (V,E) with vertices v_i and edges (i,j) i,j $\in$ {1,N}, this algorithm randomly dilutes the graph while leaving a Hamiltonian cycle subgraph intact .

INPUT: Initial adjacency matrix

OUTPUT: A sequence of edges to be diluted that will leave a subgraph of the initial set of vertices with the edges forming a Hamiltonian cycle.

START

1. *Initial step.* Randomize the sequence of numbers $L_i \in \{1,\cdots,N\}$, i = 1,2,3, ... ,N.
2. *Fixing permanent labels*
   a. Let i go from 1 to N–1 setting edge (L_i, L_{i+1}) and (L_{i+1}, L_i) to PL.
   b. Set edge (L_N, L_1) and (L_1, L_N) to PL.
3. *Dilution (this is executed exactly $N^2/3-N$ times.)*
   a. Randomly select edge (i,j).
   b. If (i,j) is not PL then set (i,j) and (j,i) to 0. Update list.
   c. Goto 3a.

END

5. Summary of Numerical Results

For each of the three dilution methods, networks of 50, 100, and 150 nodes were used in the simulations. The results are compiled from 100 runs for the 50 and 100 node cases and 50 runs for those with 150 nodes. In all cases, the critical dilution (the dilution at which the network failed to recall all of the fixed memories) is plotted for the loading ratio, $\alpha \equiv M/N$, as it is varied from 0 to about 0.25N. Figure 3 shows a typical critical dilution profile for type I dilution and N=100. The zero dilution capacity (the x-intercept) agrees with the theoretical critical capacity predicted by McEliece.

5.1. Comparison of Dilution Methods

In the cases studied, there was no significant difference in the critical dilution profile for the dilution methods used. Figure 4 shows the critical dilution profile for type I, II, and III dilution on a network with 50 nodes. The error bars show the range of one standard deviation above and below the mean for the type I dilution data set.

The fact that these different dilution methods yield very similar profiles indicates that the capacity of a diluted network is not just simply lowered as a result of a reduced effective network size. If this were the case, then type I diluted networks would have a lower critical capacity than that of a type III diluted network since type I dilution results in the random isolation of nodes, effectively lowering the network size. Additionally, these results seem to indicate that there is not a strong dependence on the arrangement

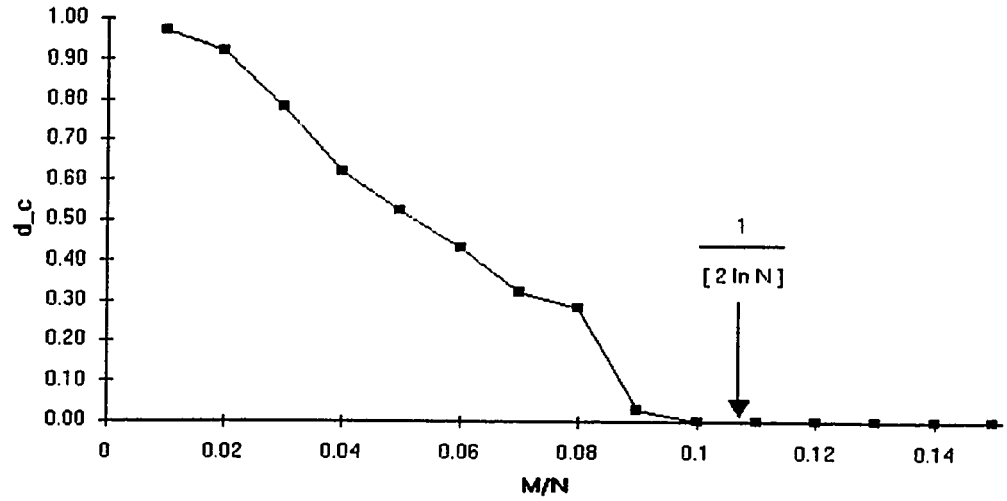

Figure 3. A typical critical dilution profile for a single run. In this case, type I dilution was used and N=100. The arrow marks the theoretical critical (undiluted) capacity.

of the connections during dilution. If this dependence were strong, then one would expect a significant difference between type I dilution (where any number of the connections to a node could be disconnected) and type III dilution (where each node has a minimum of two edges preserved at all times). In type III dilution, the remaining connections are more widely distributed among the nodes. No other attempts to control the distribution of connections were made. An experiment where the dilution is done in a way that maximizes the distribution of connections (the number of connections per node is fairly uniform) could be compared to type I dilution results to determine if there is any dependence on the arrangement of connections for a diluted network.

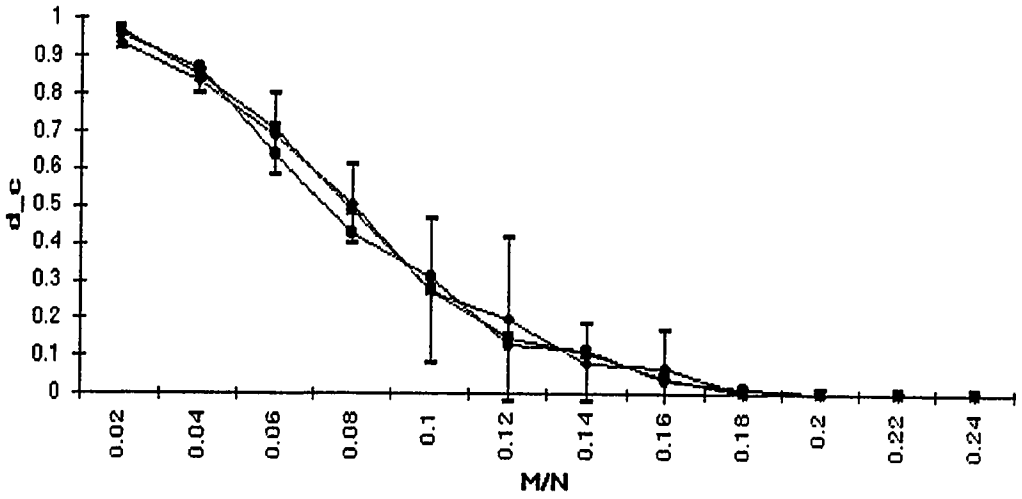

Figure 4. The critical dilution for type I, II, III dilutions. N=50. Each point is the average of ten runs. The error bars for one set is shown.

Based on the results comparing the dilution methods, it appears that the critical dilution is simply related to the average number of connections per node. It is interesting to point out here that the network structure has been shown to be an important factor when it has been diluted before learning [24].

5.2. Critical Dilution and Network Size

The critical dilution profile is slightly different for the different network sizes used in this study. This is not a real surprise since the zero-dilution critical capacity (the x-intercept) is a function of the size of the network as predicted by McEliece. Figure 5 shows the dilution profiles for networks of 50, 100, and 150 nodes subject to type I dilution. The

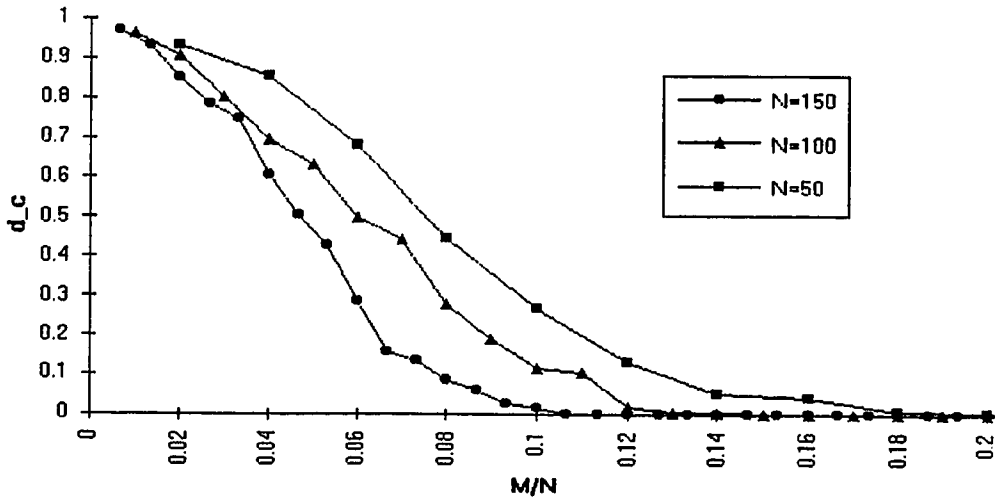

Figure 5. The critical dilution profile for networks of N = 50, 100, and 150 nodes. Each point is the average of ten runs. Data is from type I dilution.

critical dilution profile is pinned to the y-intercept of d=1.00 as it is reasonable to say that a network with no fixed patterns can suffer 100% dilution. The rest of the profile is pushed to the left as the size of the network is increased. This leads to a more linear profile. In simulations of diluted networks of 200, 400, and 800 nodes reported by Bunde [22], the maximum storage capacity as a function of the damage (dilution) is reported to be linear.

5.3. Critical Dilution and the Effective Hamming Distance

One of the main features of the dilution of networks of finite size that emerged from these experiments is the dependency of d_c on the effective hamming distance between the

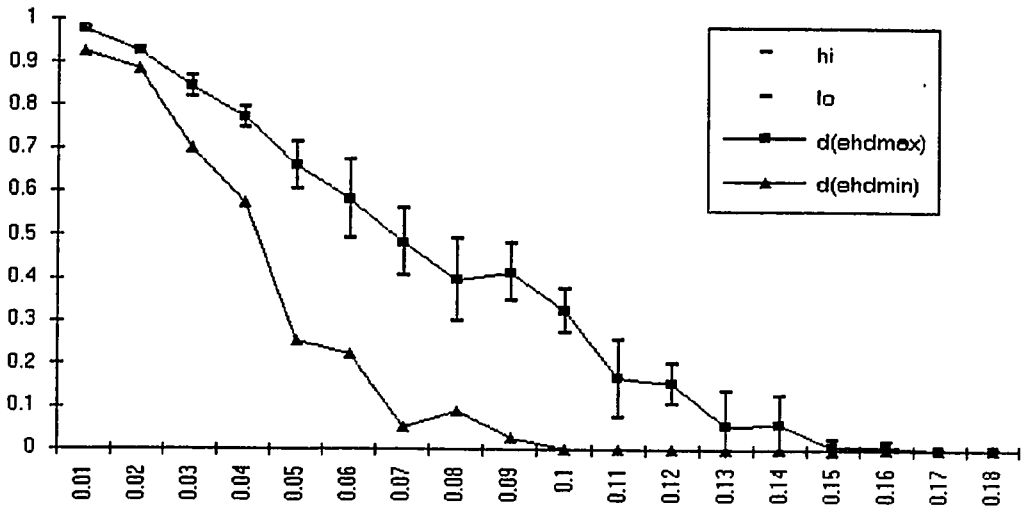

Figure 6. A comparison of critical dilution profiles for patterns with high and low average effective Hamming distance. Each point is an average of ten runs. N=100.

stored patterns. For each loading ratio, the average H_E is calculated as,

$$\langle\langle H_E \rangle\rangle = \frac{1}{M(M-1)} \sum_{\mu=1}^{M} \sum_{\substack{\nu=1 \\ \nu \neq \mu}}^{M} H_E(\xi^\mu, \xi^\nu) \quad , \tag{19}$$

where <<···>> refers to the average over all the patterns. The sensitivity to dilution is greatest for networks with memories of small << H_E >>. Figure 6 shows a critical dilution profile from the average d_c for the ten runs with the lowest << H_E >> compared to a profile generated from the ten runs with the greatest << H_E >>. The high << H_E >> was about .48N and the low << H_E >> was about .44N. The expected H_E is,

$$\langle H_E \rangle = \frac{N}{2} - \frac{Np^N}{2}\binom{N}{\frac{N}{2}} , \qquad (20)$$

where p = ½ (see Appendix B for proof). For N=100, the expected value of H_E is about 46 (.46N). This result shows that the sensitivity to dilution is strongly dependent on the average effective Hamming distance of the fixed points.

5.4. A Predictive Model Based on Numeric Results

Another prominent feature of all of the simulations is that the critical dilution as a function of the loading ratio is a not a strict linear relation as reported by Bunde. The critical dilution profile in these simulations resembles a sigmoid of the following form:

$$d_c = \frac{1}{1 - \alpha(1 - e^{(k\alpha N)})} . \qquad (21)$$

The constant k for the best fit to the average of all the runs is 0.70 for N=50 cases, 0.44 for N=100 and N=150 cases. Figure 7 shows (21) plotted with the results of the simulations for N = 50, 100, and 150 (see figures A.1, A.3, and A.5 for the individual plots of these cases). A two parameter model can be used for a better fit.

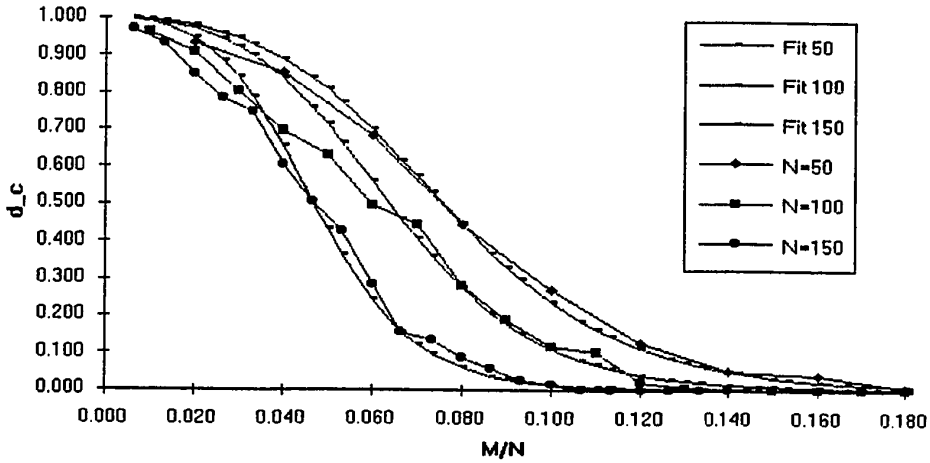

Figure 7. Comparison of numerical results with the empirical fit. The results are the average of ten runs with N=50, 100, and 150.

6. Analysis of Diluted Networks of Finite Size

6.1. Estimation of the Critical Dilution Based on Noise Analysis

In order to estimate the critical dilution of a small Hopfield neural network, we can use a signal-to-noise approach similar to that of McEliece. First, the effect of the reduced connectivity of the diluted network is considered. Next, the modified field is expressed in terms of the dilution parameter. Then the critical dilution can be determined such that the noise (the random part of the interactions) will not be greater that the signal (the interactions due to the sum-of-outer product of the original pattern).

A single node, n_i, of a fully connected network of N nodes has N–1 edges connected to it. The probability that one of the edges will become disconnected as the result of a single random cut is $2(N-1)/(N^2-N)$. The number of edges that will become disconnected as a result of c cuts will be $2c(N-1)/(N^2-N) = d(N-1)$ on average. In other words the average number of connections per node is (1–d)(N–1) after the network has been subjected to a dilution d. The field expression must now be modified by summing over the remaining connections,

$$\left\langle h_i^{(\mu)'} \right\rangle = \left\langle \sum_{\substack{j=1 \\ j \neq i}}^{N'} T_{ij} \xi_j^{(\mu)} \right\rangle = (1-d) \sum_{\substack{j=1 \\ j \neq i}}^{N} T_{ij} \xi_j^{(\mu)} \quad , \tag{22}$$

where the prime denotes that the summation is over a reduced set of edges, and $\xi^{(\mu)}$ is assumed to be one of M fixed vectors (local energy minima). Using the expression for T_{ij} defined in (7), we can express (22) as

$$\begin{aligned}\left\langle h_i^{(\mu)'} \right\rangle &= (1-d)\sum_{\substack{j=1\\ j\neq i}}^{N}\sum_{s=1}^{M} \xi_i^{(s)}\xi_j^{(s)}\xi_j^{(\mu)} \\ &= (1-d)\sum_{\substack{j=1\\ j\neq i}}^{N} \xi_i^{(\mu)}\xi_j^{(\mu)}\xi_j^{(\mu)} + (1-d)\sum_{\substack{j=1\\ j\neq i}}^{N}\sum_{\substack{s=1\\ s\neq \mu}}^{M} \xi_i^{(s)}\xi_j^{(s)}\xi_j^{(\mu)} \qquad (23) \\ &= (1-d)(N-1)\xi_i^{(\mu)} + (1-d)\sum_{j=1}^{N-1}\sum_{s=1}^{M-1} \{r\}\end{aligned}$$

where {r} is a single Bernoulli trial with $P(\xi_k^{(\mu)}) = \frac{1}{2}\delta(\xi_k^{(\mu)}-1) + \frac{1}{2}\delta(\xi_k^{(\mu)}+1)$. Therefore, the average field experienced by node i when the system is in state $\xi^{(\mu)}$ subject to dilution d is just

$$\left\langle h_i^{(\mu)'} \right\rangle = S_i^{(\mu)'} + R' \quad , \qquad (24)$$

where $S_i^{(\mu)'}$ is the signal term of magnitude (1–d)(N–1) and R is the noise term with a mean value of zero and variance (1–d)(N–1)(M–1). For large enough values of M and N, we can expect that R will have a value $-[(1-d)(N-1)(M-1)] \leq x \leq [(1-d)(N-1)(M-1)]$ with a certain probability that can be approximated by Gaussian distribution. The probability of an error is the probability that the noise is greater than the signal,

$$\epsilon = P(R \geq S) = Q(z) = \frac{1}{\sqrt{2\Pi}} \int_{z}^{\infty} e^{-\frac{t^2}{2}} dt, \qquad z = \frac{S}{\sqrt{(1-d)(N-1)(M-1)}}. \tag{25}$$

where S = (1-d)(N-1), and z is S normalized by the standard deviation. To solve this equation we can use the complementary error function, erfc(z), which can be approximated (using integration by parts)

$$erfc(y) = \frac{2}{\sqrt{\pi}} \int_{y}^{\infty} e^{-s^2} ds \approx \frac{1}{y\sqrt{\pi}} e^{-y^2} \qquad for\ N^{3/2} \gg M^{3/2} \ . \tag{26}$$

Using a change of variables ($t = \sqrt{2}\, s$) and approximating N-1 ≈ N, and M-1 ≈ M., we get

$$\epsilon = \frac{1}{2} erfc\left(\sqrt{\frac{(1-d)N}{2M}}\right) = \frac{1}{2}\sqrt{\frac{2M}{(1-d)\pi N}}\, e^{-\frac{(1-d)N}{2M}} \ . \tag{27}$$

The probability of no error in node n_i is $1-\varepsilon$. Since the error in a node is independent of the error in another node, the probability of no error in the whole network is $P_s = (1-\varepsilon)^N$. Using the lead term in the expansion of ln (1-ε) (since $\varepsilon \ll 1$),

$$\ln(P_s) = N \ln(1-\epsilon) \approx -N\epsilon = -N\sqrt{\frac{M}{2\pi N(1-d)}}\, e^{-\frac{N(1-d)}{2M}}$$

$$2\pi\, (\ln P_s)^2 = N \frac{M}{(1-d)}\, e^{-\frac{N(1-d)}{M}} \tag{28}$$

$$\ln(2\pi\, (\ln P_s)^2 = \ln MN - \ln(1-d) - \frac{N(1-d)}{M} \ .$$

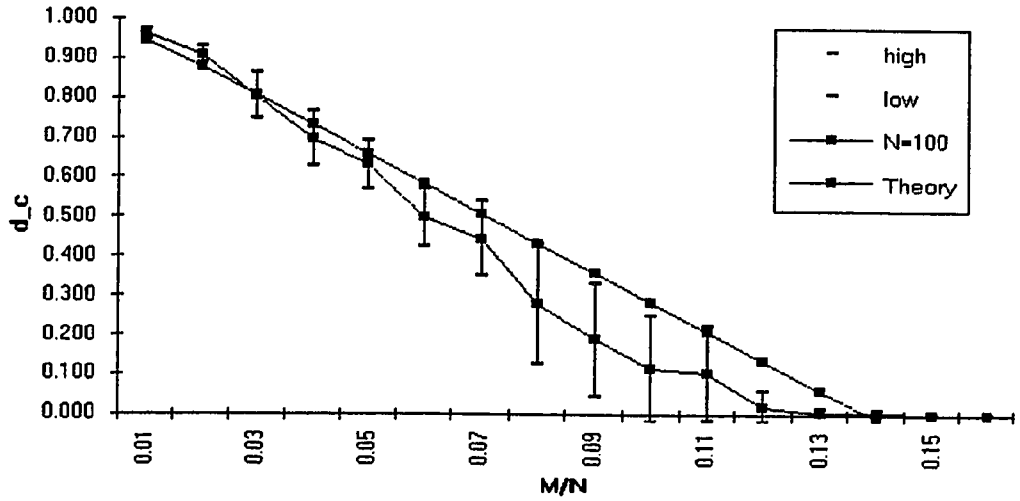

Figure 8. A comparison of the theoretical critical dilution profile with the results of the simulations for N=100. The simulation results are averaged over 10 runs.

Again, we can approximate $-\ln(1-d) = d$ on the right-hand side since $0 \le d \le 1$. In order to keep P_s fairly large, we can require that the left hand side of (28) be less than zero. This requires that,

$$d < \frac{1 - \alpha \ln(\alpha N^2)}{1 + \alpha} \quad . \tag{29}$$

This gives a reasonable estimate for the critical dilution of this network.

6.2. Comparison with Numerical Results

A comparison of the estimate for d_c (29) with the results of the simulations is shown in figure 8 (also see figures A.2, A.4, and A.6). The estimation generally falls within one

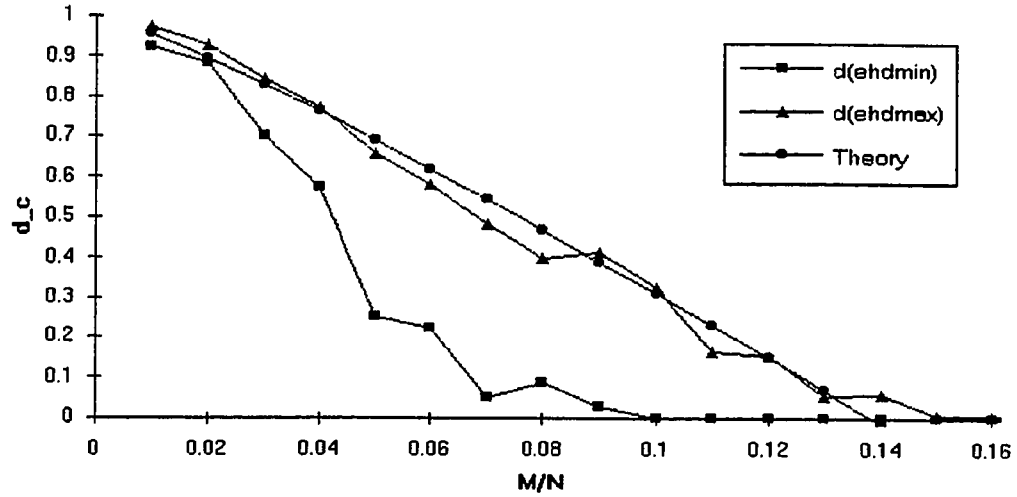

Figure 9. A comparison of the theoretical critical dilution profile with results of the simulations with high and low << H_E >>.

standard deviation from the experimental mean. If we compare (29) with the runs with high and low << H_E >>, we see that it matches with the results of runs with the higher average effective Hamming distance. This is expected since the derivation in 6.1. assumes that the patterns are *un*correlated.

This estimation is fairly linear much like the estimate derived by Sompolinsky [21]. The results shown in figures 8 and 9 even have a x-intercept near .138. This is just due to the fact that these plots are for N=100. This estimate does show the same shift in the x-intercept for increasing N, as seen in the simulation data (see figure 5).

7. Summary

Biological neural networks serve as paradigms to computational methods and architectures that may lead to extraordinary advances in the technology. We see examples of the power of these systems around us every day. The way that a common household fly is able to compute an obstacle-avoiding path in real-time as it buzzes around an annoyed person with a swatter in-hand, is an example of a computational task that cannot be done by today's fastest super-computers [10]. The way that we are able to recognize the face or the voice of a person that we have not seen in years (even though it may have changed a little) is an example of content-addressable pattern recognition. In addition, these real-world systems must function under the stresses of the environment. A certain robustness is necessary to sustain functionality in spite of damage due to natural deterioration, accident, or disease. A small degree of degradation is tolerable, but a great sensitivity to damage would be disastrous.

Neural network systems need not be large in order to be effective. The stomatogastric ganglion of the lobster, consisting of 11 neurons [25] and the sensorimotor ganglion of the leech, consisting of 8 neurons [26] are examples of small, successful neural systems in nature. Taking a minimalist approach, Randall Beer [27] demonstrated how many of the behavioral functions of *Periplaneta americana* (the American Cockroach) could be simulated by his computer version, *Periplaneta computatrix*, which uses small networks of a dozen or so neurons to do tasks such as various-gait locomotion,

edge-following, recoil, and turning. *P. computatrix* also exhibited tolerance to synaptic removal [pp. 92-108].

In this thesis, the effect of dilution on the memory capacity of a small, Hopfield-model neural network is investigated by computer simulations and analytical means. The critical dilution (the amount of dilution that a network can tolerate before it fails to recall a stored pattern) as a function of the memory loading ratio is shown to be approximately sigmoid. This is in contrast to the linear relationship reported for large networks [22]. The critical dilution at a given loading ratio is also shown to be influenced by the network size and the effective Hamming distance between all of the patterns. This again, is in contrast to the infinite network approximation where the asymptotic limit is independent of N [21] and the Hamming distance is not defined. The way in which the network is diluted does not seem be a major factor.

A noise analysis similar to that of McEliece is used to derive an estimate of the critical dilution as a function of the loading ratio and the number of nodes. Even though this analysis does not take into account the effective Hamming distances between the fixed patterns, it does predict the upper limit on the amount of dilution that a small network can tolerate. This analysis also shows the characteristic dependence on network size that is seen in the simulations of dilution in small networks.

Acknowledgement

I would like to thank Prof. Roger Dodd for all of his helpful advice throughout the many months that we worked together on this project. I would also like to thank Pat Hamill and Alex Garcia for their help and extra effort during the Spring break.

I also thank Jenet for her help with the Calcul*us*.

Appendix

A. Other Figures

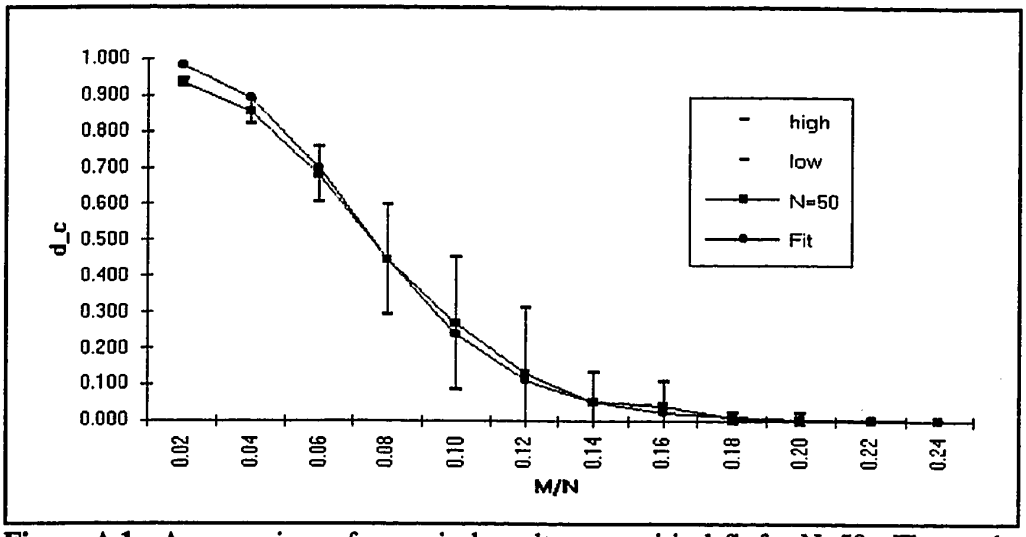

Figure A.1. A comparison of numerical results to empirical fit for N=50. The results are the average of ten runs.

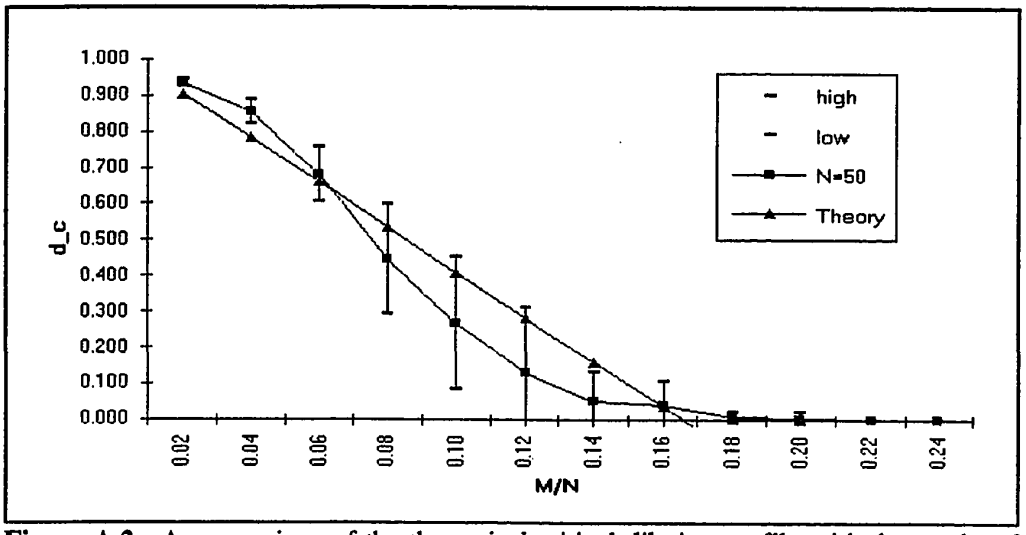

Figure A.2. A comparison of the theoretical critical dilution profile with the results of the simulations for N=50. The simulation results are the average of ten runs.

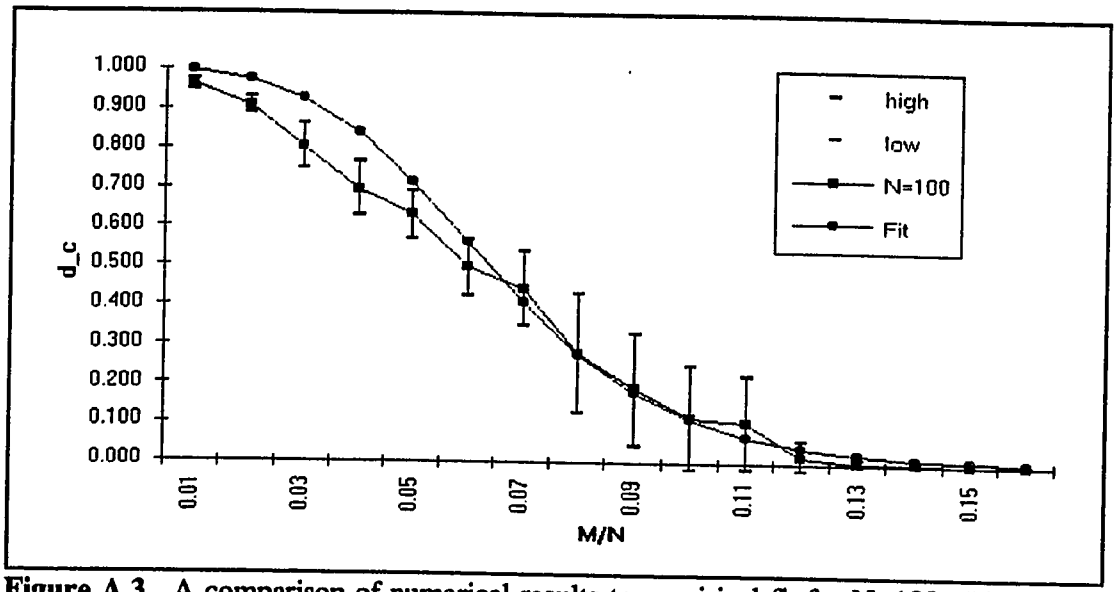

Figure A.3. A comparison of numerical results to empirical fit for N=100. The results are the average of ten runs.

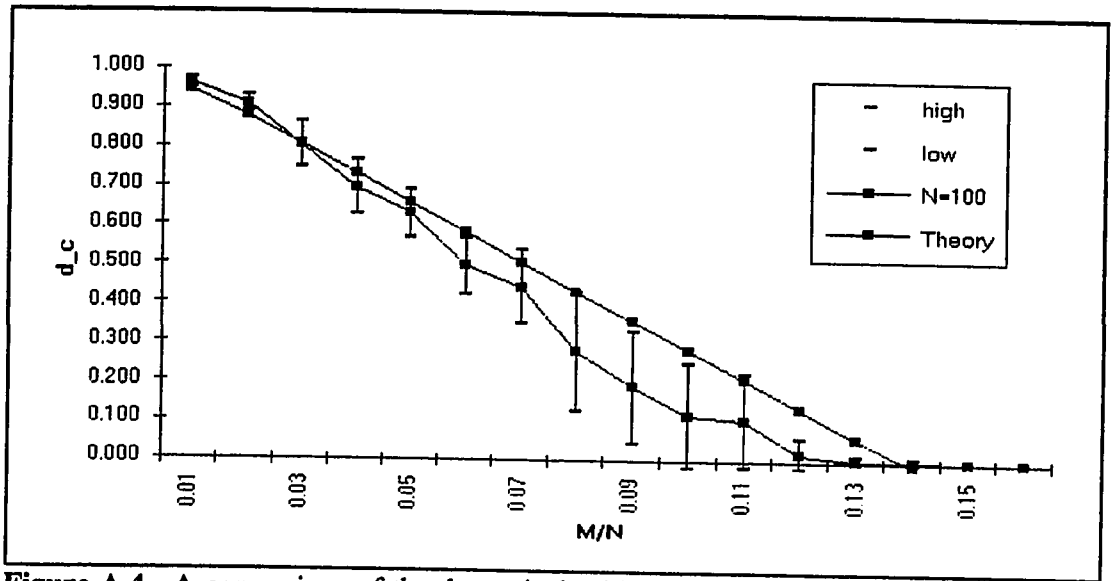

Figure A.4. A comparison of the theoretical critical dilution profile with the results of the simulations for N=100. The simulation results are the average of ten runs.

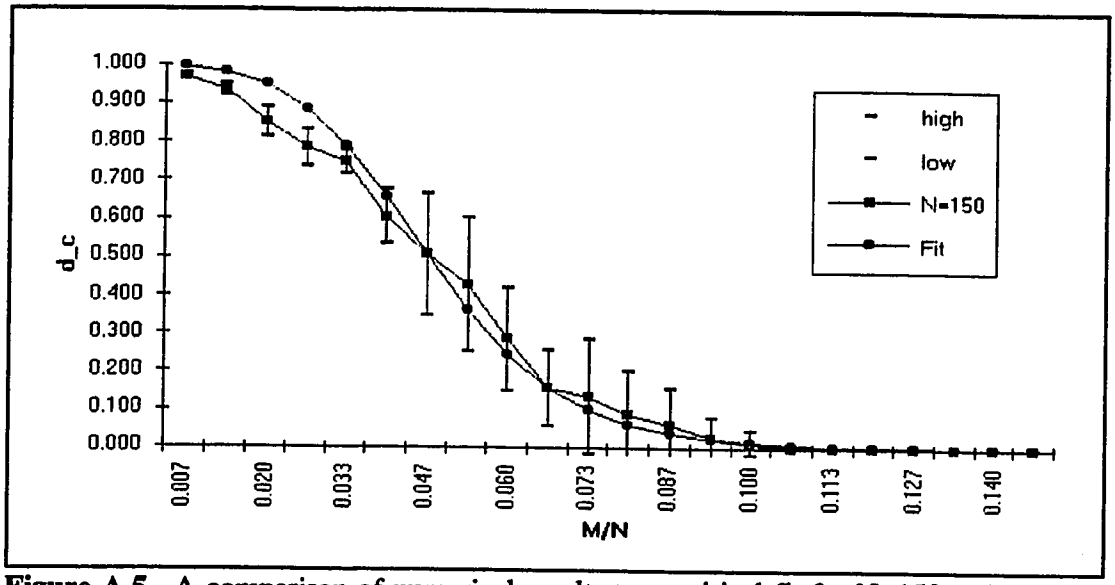

Figure A.5. A comparison of numerical results to empirical fit for N=150. The results are the average of ten runs.

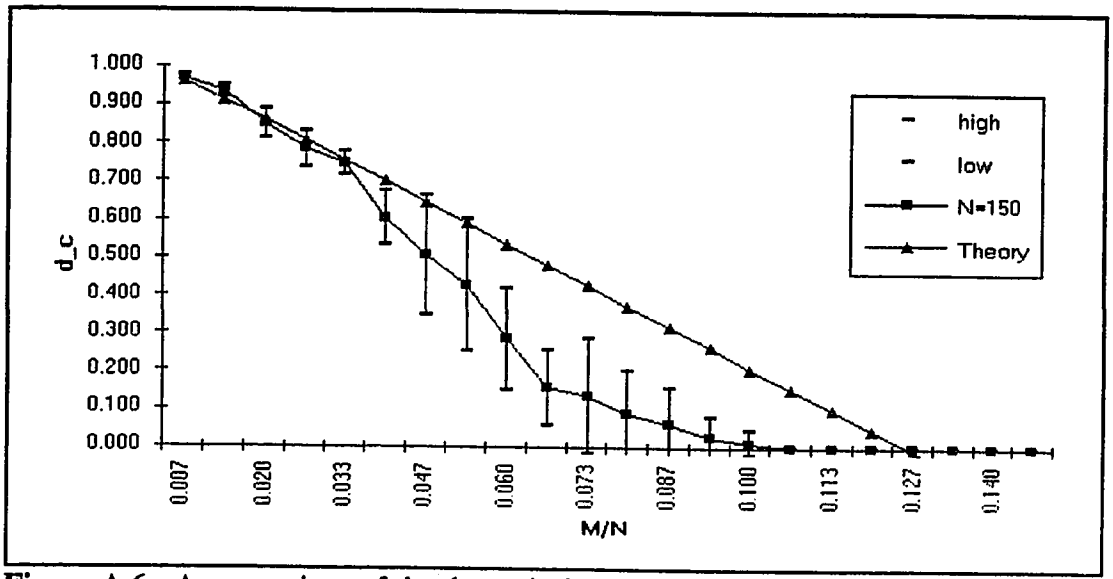

Figure A.6. A comparison of the theoretical critical dilution profile with the results of the simulations for N=150. The simulation results are the average of ten runs.

B. Proof of Equation 20

The derivation of the expectation value of the effective Hamming distance $\langle\langle H_E \rangle\rangle$ is presented here. Starting from the expression for the effective Hamming distance given in Equation 18,

$$\langle H_E \rangle = \left\langle \frac{1}{2}[N - |\langle S,T \rangle|] \right\rangle = \frac{1}{2}N - \frac{1}{2}\langle |\langle S,T \rangle| \rangle = \frac{1}{2}N - \frac{1}{2}\left\langle \left| \sum_{i=1}^{N} S_i T_i \right| \right\rangle , \tag{C.1}$$

where $\langle \blacksquare \rangle$ is the expectation value and $\langle \blacksquare, \blacksquare \rangle$ is the inner product. The probability that a node is in state $S_i = 1$ is ½. In other words, $P(S_i) = \frac{1}{2}\delta(S_i - 1) + \frac{1}{2}\delta(S_i + 1)$.

$$Let\ X_i = S_i T_i , \qquad P\{X_i = x\} = \begin{cases} k = p^2 + (1-p)^2 = \frac{1}{2} & for\ x = 1 \\ 1-k = 2p(1-p) = \frac{1}{2} & for\ x = -1 \end{cases} \tag{C.2}$$

Now consider the cumulative distribution function (cdf),

$$\begin{aligned} &Let\ S_N = \sum_{i=1}^{N} X_i , \\ &f_S(s) = P\{S_N = s\} = \binom{N}{h} k^h (1-k)^{N-h} \qquad for\ h \in I(0,\ldots,N) . \end{aligned} \tag{C.3}$$

where $s = 2h - N$.

Now consider the absolute value of the cdf.

$$
\begin{aligned}
Y_N &= |S_N| \\
f_Y(y) &= P\{S_N = y\}\,P\{S_N \geq 0\} + P\{S_N = -y\}\,P\{S_N < 0\} \\
&= \binom{N}{h} k^N\, P\left\{h \geq \frac{n}{2}\right\} + \binom{N}{N-h} k^N\, P\left\{h < \frac{n}{2}\right\}.
\end{aligned}
\tag{C.4}
$$

The expectation of Y_N can be derived as follows.

$$
\begin{aligned}
\langle Y_N \rangle &= \int_{-\infty}^{\infty} y\, f_Y(y)\,dy = \int_{0}^{\infty} y\, f_Y(y)\,dy + \int_{-\infty}^{0} -y\, f_Y(-y)\,dy \\
&= k^N \sum_{h=N/2}^{N} (2h - N)\binom{N}{h} + k^N \sum_{h=0}^{h<N/2} (N - 2h)\binom{N}{N-h}.
\end{aligned}
\tag{C.5}
$$

Using a change of variables t = N - h, and the fact that binomial(N,h) = binomial(N,t) then,

$$
\begin{aligned}
\langle Y_N \rangle &= k^N\left[2\sum_{h=N/2}^{N} h\binom{N}{h} - N\sum_{h=N/2}^{N}\binom{N}{h} + 2\sum_{t>N/2}^{N} t\binom{N}{t} - N\sum_{h=0}^{h<N/2}\binom{N}{h}\right] \\
&= -Nk^N\sum_{h=0}^{N}\binom{N}{h} + 2k^N\left[N\sum_{h=N/2}^{N}\binom{N-1}{h-1} + N\sum_{t>N/2}^{N}\binom{N-1}{t-1}.\right]
\end{aligned}
\tag{C.6}
$$

Using r=N-1, j=t-1, and the relations

$$
\sum_{h=0}^{N}\binom{N}{h} = 2^N, \qquad \text{and} \quad h\binom{N}{h} = N\binom{N-1}{h-1}, \tag{C.7}
$$

it follows that,

$$
\begin{aligned}
\langle Y_N \rangle &= -Nk^N 2^N + 2Nk^N \left[\sum_{t=0}^{N/2} \binom{r}{t} + \sum_{j=N/2}^{r} \binom{r}{j} \right] \\
&= -N + 2Nk^N \left[2^{N-1} + \binom{N-1}{N/2} \right] \\
&= -N + N + 2Nk^N \binom{N-1}{N/2} \\
&= Nk^N \binom{N}{N/2}.
\end{aligned}
\tag{C.8}
$$

Using this and (C.1), the expected effective Hamming distance is

$$
\langle H_E \rangle = = \frac{1}{2}N - \frac{1}{2}\langle Y_N \rangle = \frac{N}{2} - \frac{Nk^N}{2}\binom{N}{\frac{N}{2}}. \tag{C.9}
$$

C. Program Descriptions and Codes

Note on Random Number Generator

The random number generators used in this program are from *Numerical Recipes in C*. The routine used to generate the random binary patterns is from pp. 225. This routine returns a random integer equal to zero or one with probability 1/2. The routine used to generate the cut sequence is from pp. 213. This routine returns a floating point value uniformly distributed between 0 and 1.

Code used for Simple dilution

```
/*************************************************************************
HOP10.C

      * random disconnection without constraint.
      * calculates total effective hamming distances.
      * critical dilution is at first memory loss.
06-Nov-1991
*************************************************************************/

#include <stdio.h>

#define N 100                          /* number of bits in each pattern */

void init(void), new_pats(void), cut_order(void), clear_mem(void);
void store_pattern(int), terminate(void), exit(int);
float ran3(unsigned long *);
unsigned int sum_hamming_dist(int);
unsigned long seed;
int probe_pattern(int), sign(int), irbit(unsigned long *);
int DMAX = (N*N-N)/2, w[N][N], pat[N][N], cut[(N*N-N)/2][2];
FILE *fp;

void main()
{
      int k, r, mem, m, REP;
      unsigned int totehd;
      printf("Enter number of reps : ");scanf("%d", &REP);
      init();
      for(r = 1; r < REP+1; r++)
      {
```

```
            printf("(%d) ", r);
            new_pats();
            for(mem = 1; mem < (int)(.26 * N); mem++)
            {
                  /*****   Determine order of dilution   *****/
                  printf("(%d) ", r);
                  cut_order();

                  /*****   Clears the weight matrix   *****/
                  clear_mem();

                  /*****   Stores (mem) patterns   *****/
                  store_pattern(mem);

                  /*****   Dilution Loop   *****/
                  printf("Probing memories and diluting ... ");
                  for(k=1; k<DMAX+1; k++)
                  {
                        /*****   Probes (mem) patterns   *****/
                        for(m = 1; m < mem+1; m++)
                        {
                              if (probe_pattern(m) != N) goto report;
                        }
                        w[ cut[k][1] ][ cut[k][2] ] = 0;
                        w[ cut[k][2] ][ cut[k][1] ] = 0;
                  }
                  report:
                        totehd = sum_hamming_dist(mem);
                        fprintf(fp,"%d, %d, %u, %d\n", N, mem,
                                    totehd, k-1);
                  printf("done!\n");
            };
      fflush(fp);
      };
      terminate();
}

void init(void)
/*************************************************************************
INITIALIZATION ROUTINE.
      * initializes stfile
      * gets an integer seed for the random number generator
*************************************************************************/
{
      if ((fp = fopen("hop10b.dat", "a+")) == NULL) exit(1);
      printf("Enter an integer seed : "); scanf("%lu", &seed);
      fprintf(fp, "From HOP10.c, random dilution without constraint.\n");
      fprintf(fp, "N = %d\n", N);
      fprintf(fp, "seed = %lu\n", seed);
      fprintf(fp, "N,  M, totehd/2, cuts \n");
}

void new_pats(void)
/*************************************************************************
PATTERN INITIALIZATION ROUTINE.
      * creates an N x N matrix of random patterns for testing
            (N patterns of N bits)
      * initializes the weight matrix
```

```
      * pat[m][k] is the kth bit of memory m
*****************************************************************************/
{
      int m,k;
      printf("Initializing random patterns ... ");
      for(m = 1; m < N+1; m++)
      {
            for(k = 0; k< N; k++)
            {
                  pat[m][k] = irbit(&seed) * 2 - 1;
            }
      }
      printf("done!\n");
}

void cut_order(void)
/*****************************************************************************
DETERMINE ORDER OF DILUTION
*****************************************************************************/
{
      int i, j, k;
      printf("   Randomizing dilution sequence ... ");
      for(i = 0; i < N; i++)
      {
            for(j = 0; j < N; j++) w[i][j] = 1;
            w[i][i] = 0;
      }
      for (k=1; k < DMAX+1; k++)
      {
            do
            {
                  i = (int) (ran3(&seed) * N);
                  j = (int) (ran3(&seed) * N);
            } while (!w[i][j]);
            w[i][j] = 0; w[j][i] = 0;
            cut[k][1] = i; cut[k][2] = j;
      }
      printf("done!\n");
}

void clear_mem(void)                  /* tabula rasa */
/*****************************************************************************
CLEAR MEMORY
      * sets the weight matrix to zero
*****************************************************************************/
{
      int i,j;
      printf("  Clearing memory ... ");
      for(i = 0; i < N; i++)
      {
            for(j = 0; j < N; j++)  w[i][j] = 0;
      }
      printf("done!\n");
}

void store_pattern(int mem)
/*****************************************************************************
PATTERN STORING ROUTINE
```

```
      * stores the (mem) patterns in the weight matrix using Hebb rule
      * sets the diagonal of the weight matrix to zero
*************************************************************************/
{
      int m, i, j;
      printf(" Storing %d patterns ... ", mem);
      for(m = 1; m < mem+1; m++)
      {
            for(i = 0; i < N; i++)
            {
                  for(j = 0; j < N; j++)
                  {
                        w[i][j] += pat[m][i] * pat[m][j];
                  }
                  w[i][i] = 0;
            }
      }
      printf("done!\n");
}

probe_pattern(int m)
/*************************************************************************
PROBING AND TESTING ROUTINE
      * probes the memory with the mth pattern
      * compares each output bit with the probe
      * returns correlation between input and output
*************************************************************************/
{
      int i, j, outbit, corr = 0;
      for(i = 0; i < N; i++)
      {
            outbit = 0;
            for(j = 0; j< N; j++) outbit += w[i][j] * pat[m][j];
            corr += pat[m][i]*sign(outbit);
            /* if(pat[m][i] == sign(outbit)) corr++*/
      }
      return corr;
}

unsigned int sum_hamming_dist(int mem)
/*************************************************************************
CALCULATES TOTAL EFFECTIVE HAMMING DISTANCES FOR MEMORIES 1 TO mem.
*************************************************************************/
{
      int m1, m2, k, ehd;
      unsigned int totehd = 0;
      for(m1 = 1; m1 < mem+1; m1++)
      {
            for(m2 = 1; m2 < m1+1; m2++)
            {
                  ehd = 0;
                  for(k = 0; k < N; k++) if(pat[m1][k] != pat[m2][k])
ehd++;
                  if (ehd > N/2) ehd = N-ehd;    /* this yields ehd */
                  totehd += ehd;
            };
      }
      return totehd;
```

```
}

int sign(int x)
/*************************************************************************
SIGN(X) ROUTINE
      * returns 1 if x > 0
      * returns -1 if x = 0 or x < 0    (arbitrary convention)
*************************************************************************/
{
      if(x > 0) return(1);
      else return(-1);
}

void terminate(void)
/*************************************************************************
TERMINATES STFILE AND PROGRAM
*************************************************************************/
{
      if(fclose(fp) != 0) fprintf(stderr, "Error closing file\n");
      exit(0);
}

#define IB1 1
#define IB2 2
#define IB5 16
#define IB18 131072

int irbit(unsigned long *iseed)
/*************************************************************************
RANDOM BIT ROUTINE (from 'NUMERICAL RECIPES in C' pp. 224)
      * takes an integer seed
      * returns a random bit {0,1}
      * randomizes iseed
*************************************************************************/
{
        unsigned long newbit;
        newbit = (*iseed & IB18) >> 17
                ^ (*iseed & IB5) >> 4
                ^ (*iseed & IB2) >> 1
                ^ (*iseed & IB1);
        *iseed=(*iseed << 1) | newbit;
        return (int) newbit;
}

#define MBIG 1000000000
#define MZ 0
#define FAC (1.0/MBIG)
#define MSEED 161803398

float ran3(unsigned long *idum)
/*************************************************************************
RANDOM NUMBER ROUTINE (from 'NUMERICAL RECIPES in C' pp. 224)
*************************************************************************/
{
      static int inext, inextp;
      static long ma[56];
      static int iff=0;
      long mj, mk;
```

```
        int i, ii, k;
        if(*idum < 0 || iff == 0) {
              iff=1;
              mj = MSEED-(*idum < 0 ? -*idum : *idum);
              mj %= MBIG;
              ma[55]=mj;
              mk=1;
              for (i=1; i<=54;i++) {
                    ii=(21*i) % 55;
                    ma[ii]=mk;
                    mk=mj-mk;
                    if (mk < MZ) mk += MBIG;
                    mj=ma[ii];
              }
              for(k=1;k<=4;k++)
                    for (i=1; i<=55; i++) {
                          ma[i] -= ma[1+(i+30) % 55];
                          if (ma[i] < MZ) ma[i] += MBIG;
              }
              inext=0;
              inextp=31;
              *idum=1;
        }
        if (++inext == 56) inext=1;
        if (++inextp == 56) inextp=1;
        mj=ma[inext]-ma[inextp];
        if (mj < MZ) mj += MBIG;
        ma[inext]=mj;
        return mj*FAC;
}
```

References

[1] Huang K 1987 *Statistical Mechanics* (New York: Wiley) pp 341-63

[2] Frisch U, Hasslacher B, and Pomeau Y 1986 Lattice-gas automata for the Navier-Stokes equation *Phys. Rev. Lett.* **56** 1505-08

[3] Hiebeler D and Tatar R 1992 Cellular automata and discrete physics *Introduction to Nonlinear Physics* ed L Lam (New York: Springer-Verlag)

[4] Margolus N 1984 Physics-like models of computation *Physica* **10D** 81-95

[5] Wolfram S (ed) 1986 *Theory and Applications of Cellular Automata* (Singapore: World Scientific)

[6] Toffoli T and Margolus N 1989 *Cellular Automata Machines* (Cambridge, MA: MIT Press)

[7] McCulloch W S and Pitts W 1943 A logical calculus of the ideas immanent in nervous activity *Bul. of Math. Biophys.* **5** 115-33

[8] Hebb D O 1949 *The Organization of Behavior* (New York: Wiley)

[9] Minsky M and Papert S 1969 *Perceptrons* (Cambridge MA: MIT Press)

[10] Rumelhard D E, McClelland J L, et al. 1986 Learning internal representation by error propagation *Parallel Distributed Processing* vols 1 (Cambridge, MA: MIT Press) pp 318-364

[11] Anderson J A 1972 A simple neural network generating an interactive memory *Math. BioSci.* **14** 197-220

[12] Muller B and Reinhard J 1990 *Neural Networks An Introduction* (Berlin: Springer-Verlag) pp 12-22

[13] Hopfield J J 1982 Neural networks and physical systems with emergent collective computational abilities *Proc. Natl. Acad. Sci.* USA **79** 2554-8

[14] Amit D J, Gutfreund H, et al. 1985 Spin-glass models of neural networks *Phys. Rev. A* **55** 1007-18

[15] Kanter I and Sompolinsky H 1987 Associative recall of memory without errors *Phys. Rev. A* **35** 380

[16] Gardner E 1988 The space of interactions in neural network models *J. Phys. A* **21** 257-70

[17] McEliece R J, Posner E C, et al. 1987 The Capacity of the Hopfield associative memory *IEEE Trans. Inform. Theory* **IT-33** 461-482

[18] Derrida B, Gardner E, et al. 1987 An exactly solvable asymmetric neural network model *Europhys. Lett* **4** 167-

[19] Kree R and Zippelius A 1991 Asymmetrically diluted neural networks *Models of Neural Networks* ed E Domany et al. (Berlin: Springer-Verlag) pp. 193-212

[20] Bouten M 1990 Storage capacity of diluted networks *Statistical Mechanics of Neural Networks* ed L Garrido (Berlin: Springer-Verlag) pp 225-236

[21] Sompolinsky H 1986 Neural networks with nonlinear synapses and a static noise *Phys. Rev. A* **34** 2571-4

[22] Koscielny-Bunde E 1990 Pattern recognition in damaged Hopfield networks *Parallel Processing in Neural Systems and Computers* ed R Eckmiller et al. (N. Holland: Elsevier Science Publishers) pp. 459-462

[23] Dasgupta S, Ghosh A, et al. 1989 Convergence in neural memories *IEEE Trans. Inform. Theory* **IT-35** 1069-72

[24] Yanai H-F, Sawada Y, et al. 1991 Dynamics of an auto-associative neural network model with arbitrary connectivity and noise in the threshold *Network* **2** 295-314

[25] Tsung F-S, Cottrell W, et al. 1990 Some experiments on learning stable network oscillations *Proceedings of International Joint Conference on Neural Networks 1990* **I** 169-74

[26] Lockery S, Fang Y, et al. 1990 A dynamical neural network model of sensorimotor transformations in the leech *Proceedings of International Joint Conference on Neural Networks 1990* I 183-88

[27] Beer R 1990 *Intelligence as Adaptive Behavior: An Experiment in Computational Neuroethology* (San Diego: Academic Press)